# The Role of Limitations and SLAs in the API Industry


Antonio Gamez-Diaz
Universidad de Sevilla
Seville, Spain
antoniogamez@us.es

Pablo Fernandez
Universidad de Sevilla
Seville, Spain
pablofm@us.es

Antonio Ruiz-Cortés
Universidad de Sevilla
Seville, Spain
aruiz@us.es

Pedro J. Molina
Metadev
Seville, Spain
pjmolina@metadev.pro

Nikhil Kolekar
PayPal
San Jose, California, USA
nikhil@openweave.ai

Prithpal Bhogill
Google
Mountain View, California, USA
prithpal@google.com

Madhurranjan Mohaan
Google
Mountain View, California, USA
madhurranjanm@google.com

Francisco Méndez
AsyncAPI Initiative
Barcelona, Spain
fmvilas@gmail.com



## ABSTRACT

As software architecture design is evolving to a microservice paradigm, RESTful APIs are being established as the preferred choice to build applications. In such a scenario, there is a shift towards a growing market of APIs where providers offer different service levels with tailored limitations typically based on the cost.

In this context, while there are well established standards to describe the functional elements of APIs (such as the OpenAPI Specification), having a standard model for Service Level Agreements (SLAs) for APIs may boost an open ecosystem of tools that would represent an improvement for the industry by automating certain tasks during the development such as: SLA-aware scaffolding, SLA-aware testing, or SLA-aware requesters.

Unfortunately, despite there have been several proposals to describe SLAs for software in general and web services in particular during the past decades, there is an actual lack of a widely used standard due to the complex landscape of concepts surrounding the notion of SLAs and the multiple perspectives that can be addressed.

In this paper, we aim to analyze the landscape for SLAs for APIs in two different directions: i) Clarifying the SLA-driven API development lifecycle: its activities and participants; 2) Developing a catalog of relevant concepts and an ulterior prioritization based on different perspectives from both Industry and Academia. As a main result, we present a scored list of concepts that paves the way to establish a concrete road-map for a standard industry-aligned specification to describe SLAs in APIs.



## CCS CONCEPTS

**• Information systems → RESTful web services**; **• Software and its engineering → Extra-functional properties**; **System description languages**.

## KEYWORDS

RESTful APIs, SLA, OpenAPI Specification, SLA-driven APIs, API Gateways

**ACM Reference Format:**

Antonio Gamez-Diaz, Pablo Fernandez, Antonio Ruiz-Cortés, Pedro J. Molina, Nikhil Kolekar, Prithpal Bhogill, Madhurranjan Mohaan, and Francisco Méndez. 2019. The Role of Limitations and SLAs in the API Industry. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3338906.3340445


## 1 INTRODUCTION

In the last decade, RESTful APIs are becoming a clear trend as composable elements that can be used to build and integrate software [6, 12]. One of the key benefits this paradigm offers is a systematic approach to information modeling leveraged by a growing set of standardized tooling stack. In this context, the term of *API Economy* is being increasingly used to describe the movement of the industries to share their internal business assets as APIs [22] not only across internal organizational units but also to external third parties; in doing so, this trend has the potential of unlocking additional business value through the creation of new assets [4]. In fact, we can find a number of examples in the industry that are deployed solely as APIs (such as Meaningcloud[1], Flightstats[2] or Twilio[3]).

In order to be competitive in this such a growing market of APIs, at least two key aspects can be identified: i) *ease of use* for its potential developers; ii) a flexible usage *plan* that fits their customer's demands.

Regarding the *ease of use* perspective, third-party developers need to understand how to use the exposed APIs so it becomes necessary to provide good training material but, unfortunately, several API providers do not often write good documentation of their products [7]. Notwithstanding, during the last years, the *OpenAPI*


*ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia*



[1]https://www.meaningcloud.com/products/pricing
[2]https://developer.flightstats.com/getting-started/pricing
[3]https://www.twilio.com/sms/pricing

*Specification*[4] (OAS), formerly known as *Swagger* specification, has become the *de facto* standard to describe RESTful APIs from a functional perspective providing an ecosystem that helps the developer in several aspects of the API development lifecycle[5].

The benefits are twofold: from the API provider's perspective, there are tools aimed to automate the server scaffolding, an interactive documentation portal creation or the generation of unit test cases; from API consumer's perspective, there are tools to automate the creation of API clients, the security configuration or the endpoints discovery and usage [1, 19, 21].

Concerning the usage *plans* perspective, as APIs are deployed and used in real settings, the need for non-functional aspects is becoming crucial. In particular, the adoption of Service Level Agreements (SLAs) [17] could be highly valuable to address significant challenges that industry is facing, as they provide an explicit placeholder to state the guarantees and limitations that a provider offers to its consumers. Exemplary, these limitations (such as *quotas* or *rates*) are present in most common industrial APIs [8] and both API providers and consumers need to handle how they monitor, enforce or respect them with the consequent impact in the API deployment/consumption.

However, to the best of our knowledge, there is no widely accepted model to describe usage plans including elements such as cost, functionality restrictions or limits. In this context, a new type of infrastructure, coined as API Gateway [10], has emerged to support API developers in the management of multiple non-functional aspects such as consumer authentication, request throttling or billing. From a deployment perspective, API Gateways are usually implemented as virtual appliances, virtual machine images or reverse proxies that promote a decoupling from the main API artifact. In contrast, the vendor-specific approach to non-functional concerns typically represents a strong dependence with the API Gateway provider.

In this paper, we aim to analyze the landscape in the SLA and limitations for APIs directly from those participants who have shown interest on participating in the definition of an industrial standard for SLAs in APIs. Specifically, we have started up conversations with members of the OpenAPI Initiative who belong to the SLA interest group aiming to gather information about their industrial perspective of the role of SLAs and limitations in the APIs.

The rest of the paper is structured as follows: in Section 2 we introduce, briefly, the idea of Service Level Agreements (SLA) and its importance in the API ecosystem. Next, in Section 3, we describe the related work. Continuing, in Section 4 we describe the SLA-driven API lifecycle. Further, in Section 5 we present the industrial insights from different participants. Finally, in Section 6, we show some final remarks and conclusions.

## 2 SLAS IN A NUTSHELL

Service Level Agreements (SLAs) consist of a set of terms that include information about functional features, non-functional guarantees, compensation, termination terms and any other terms with relevant information to the agreement. An agreement signed by all interested parties should be redacted carefully because a failure to specify their terms could carry penalties to the initiating or responding party. Therefore, agreement terms should be specified in a consistent way, avoiding contradictions between them. However, depending on the complexity of the agreement, this may become a challenging task. SLAs can, therefore, be used to describe the rights and obligations of parties involved in the transactions (typically the service consumer and the service provider); among other information, SLA could define guarantees associated with the idea of Service Level Objectives (SLOs) that normally represent key performance indicators of either the consumer or the provider. In case the guarantee is under-fulfilled or over-fulfilled SLAs could also define some compensations (i.e. penalties or rewards). In such a context, during the last years, there have been important steps towards the automation of the management of SLAs, however, the formalization in SLAs still remains an important challenge.

A *SLA* typically contains these concepts:

**Name** identifies the agreement and can be used for reference.

**Context** includes information such as the name of the parties and their roles as initiator or responder in the agreement. Additionally, it can include other important information for the agreement.

**Terms** the two main types of terms are:

- **Service terms** they provide service information by means of:
  - **Service description terms** which includes information to instantiate or identify the services and operations involved in the agreement.
  - **Service properties** which includes the measurable properties that are used in expressing guarantee terms. They consist of a set of variables whose values can be established inside the service description term. These terms play an key role in the definition of the service level which is actually offered to clients and the price they pay for. For instance, in APIs, it is common to see quota (e.g., 30K request/month) and rate (e.g., 1 request/second) limitations that define the service.
- **Guarantee terms** they describe the service level objectives (*SLOs*) agreed by a specific obligated party, using Service Level Indicators (*SLIs*), a set of carefully defined quantitative measures of some aspect of the level of service that is provided. It also includes the scope of the term (e.g. if it applies to a certain operation of a service or the whole service itself) and a qualifying condition that specifies the validity condition under which the term is applied. Guarantee terms often include compensations [17], that is, penalties (or rewards) applied when the SLO is unfulfilled or overfulfilled.

The concept of SLA is, very frequently, misunderstood: some services claim to have an SLA when they are only defining the service description terms (e.g., limitations). SLAs are agreements, that is, an explicit or implicit contract with your users that includes consequences of the meeting (or missing) the SLOs they contain [3, 20]. In many services, including APIs, there is no SLA: if nothing happens if the SLOs are not being met, it is not an SLA, but a mere description of SLOs and service properties.

[4] The latest version of the OpenAPI Specification is available at https://github.com/OAI/OpenAPI-Specification
[5] https://openapi.tools

In the industry, the way in which a customer can select and purchase a certain service level is by using pricing plans. In Figure 1 it is depicted a real plan extracted from *FullContact*[6], a product which includes an API for managing and organizing contacts in a collaborative way and it also matches emails addresses looking for publicly available information on the Internet to enrich the profiles.

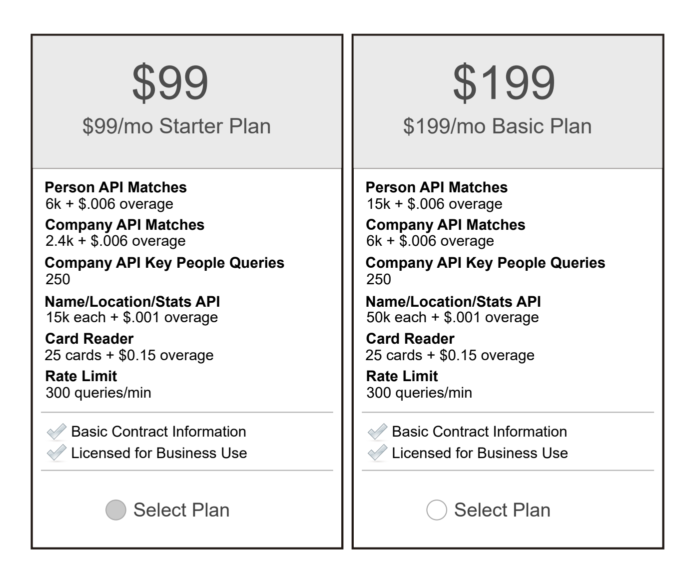



**Figure 1: Example of an API plan**

This example is composed of two paid plans having a fixed price that is monthly billed. Regarding the *limitations*, for each resource, a *quota* is being applied; for instance, in the *starter* plan, only 6000 matches over Person are available. Nevertheless, an overage is defined, that is, it is possible to overcome the limit by paying a certain amount of money; in this case, $0.006 per each request. Regardless of the accessed resources, a common *rate* of 300 queries per minute is being applied.

In this example, there is neither guarantee term nor SLOs. All these elements belong to the set of service properties, particularly, the limitations, which are, actually, defining the service level (e.g., *free*, *starter* or *basic*)

## 3 RELATED WORK

The software industry has embraced integration as a key challenge that should be addressed in multiple scenarios. In such a context, the proliferation of APIs is a reality that has been formally analyzed: in [18], authors performed an analysis of more than 500 publicly-available APIs to identify the different trends in current industrial landscape with the following key results: in terms of paradigm they conclude that 500 out of 522 analyzed APIs provide an API based on REST; regarding the *format*, the authors identified that nearly two thirds of the APIs support JSON without supporting XML. Concerning the *access control*, authors showed that most APIs require some form of service registration for developers to start using the API. Regarding the *documentation*, they showed that generated documentation is being used in about a half of the APIs, with documents generated by SwaggerUI (from an OpenAPI Specification) taking the lead, suggesting some tendency to make the API documentation machine-readable and understandable as well. Specifically, from a functional point of view, there is a clear trend with respect to the functional description of the service: during the last years, the OpenAPI Specification has consolidated as a *de-facto* standard to define the different functional properties an API provides. One of the reasons behind this success has been a growing ecosystem of tools that leverages from the API development life-cycle based on the information included in OAS: from automated code generators that create an initial scaffolding of the API to dynamic documentation portals that allow developers to understand and test the API usage.

[6]https://www.fullcontact.com/developer

**Table 1: Analysis of SLA Models**

| Name | F1 | F2 | F3 | F4 | F5 | F6 | F7 |
|---|---|---|---|---|---|---|---|
| **SLAC** [24] | DSL | | | | | ✓ | ✓ |
| **CSLA** [14] | XML | | ✓ | | | ✓ | |
| **L-USDL Ag.** [11] | RDF | ✓ | ✓ | | † | ✓ | |
| **rSLA** [23] | Ruby | ✓ | | ✓ | ✓ | | ✓ |
| **SLAng** [15] | XML | ✓ | | | | | |
| **WSLA** [16] | XML | ✓ | ✓ | | ✓ | | |
| **SLA*** [13] | XML | ✓ | ✓ | | ✓ | | |
| **WS-Ag.** [2] | XML | ✓ | ✓ | ✓ | † | | |

† Supported with minor enhancements or modifications.

In such a consolidated market of APIs, non-functional aspects are also becoming a key element in the current landscape. In [8], authors analyze a set of the 69 real APIs in the industry to characterize the variability in its offerings, obtaining a number of valuable conclusions about real-world APIs, such as: (i) Most APIs provide different capabilities depending on the tier or plan of the API consumer is willing to pay. (ii) Usage limitations are a common aspect all APIs describe in their offerings. (iii) Limitations over API requests are the most common including quotas over static periods of times (e.g., *1.000 request each natural day*) and rates for dynamic periods of times (*3 request per second*). (iv) Offerings can include a wide number of metrics over other aspects of the API that can be domain-independent (such as the number of returned results or the size in bytes of the request) or domain-dependent (such as the CPU/RAM consumption during the request processing or the number of different resource types). Based on these conclusions, we identify the need for non-functional support in the API development life-cycle and the high level of expressiveness present in the API offerings.

Furthermore, as monitoring is a key aspect, a number of works have been presented aiming to analyze different approaches for runtime monitoring. In [20], authors developed a comparison framework for runtime monitoring approaches and validate it by applying it to 32 existing approaches and by comparing 3 selected approaches in the light of different monitoring scenarios.

Furthermore, during the last decade, a number of SLA models have been presented. We have analyzed the most prominent academic and industrial proposals aimed to the definition of SLAs in both traditional web services and cloud scenarios.

Specifically, in Table 1, we have considered 7 aspects to analyze in each SLA proposal, namely: **F1** determines the format in which the document is written syntax; **F2** shows whether the target domain is web services; **F3** indicates if it can model more than one offering (i.e., different operations of a web service); **F4** determines if it allows modeling hierarchical models or overriding properties and metrics; **F5** shows whether temporal concerns can be model (e.g., in metrics); **F6** indicates if there exists a tool for assisting users to model using this proposal; **F7** determines if there exists a tool/framework for enacting the SLA.

Based on the comparison of the different SLA models (summarized in Table 1), we highlight the following conclusions: (i) None of the specifications provides any support or alignment with the OpenAPI Specification; (ii) Most of the approaches provide a concrete syntax on XML, RDF (some of them they even lack concrete syntax) and there is no explicit support to YAML or JSON serializations. (iii) An important number of proposals are complete, but others leave some parts open to being implemented by practitioners. (iv) Besides the fact that a number of proposals are aimed to model web services, they are focused on traditional SOAP web services rather than RESTful APIs. In this context, they do not address the modeling standardization of the RESTful approach: i.e., the concept of a resource is well unified (a URL), and the amount of operations is limited (to the HTTP methods, such as GET, POST, PUT and DELETE). This lack of support of the RESTful modeling prevents the approaches to have a concise and compact binding between functional and non-functional aspects. (v) They do not have enough expressiveness to model limitations such as quotas and rates, for each resource and method and with complete management of temporally (static/sliding time windows and periodicity) present in the typical industrial API SLAs. (vi) Most proposals are designed to model a single offering and they mostly lack support to modeling hierarchical models or overriding properties and metrics (F4); in such a context, they cannot model a set of tiers or plans that yield a complex offering that maintains the coherence by model and instead they rely on a manual process that is typically error-prone. (vii) finally, the ecosystem of tools proposed in each approach (in the case of its existence) is extremely limited and aimed to be solely as a prototype; moreover, they apparently are not integrated into a developer community nor there is evidence of this usage by practitioners in the industry.

## 4 INTRODUCING SLAS IN THE API LIFECYCLE

In spite of the fact that each organization could address the API lifecycle with slightly different approaches, we identify a minimal set of general stages and activities. The first activity corresponds with the actual *Functional Development* of the API implementing and testing the logic; next a *Deployment* activity where the developed artifact is configured to be executed in a given infrastructure; finally, once the API is up and running, an *Operation* activity starts where the requests from consumers can be accepted. This process is a simplification that can be evolved to add intermediate steps (such as testing) or to include an evolutionary cycle where different versions are deployed progressively. In order to incorporate SLAs in this process, we expand to this basic lifecycle where both API Provider and API Consumer interact (as depicted in Figure 2).

Specifically, from the provider's perspective, the *Functional Development* can be developed in parallel with a *SLA modeling* where the actual SLA offering is written and stored in a given *SLA Registry*. Once both the functional development and the SLA modeling has concluded, the *SLA instrumentation* must be carried out, where the tools and/or developed artifacts are parameterized, so they can adjust their behavior depending on a concrete SLA and provide the appropriate metrics to analyze the SLA status. Next, while the *deployment* of the API takes place, a parallel activity of *SLA enactment* is developed where the deployment infrastructure should be configured in order to be able to enforce the SLA before the API reaches the *operation* activity.

Complementary, from consumer's perspective, once the provider has published the SLA offering (i.e., *Plans*) in the *SLA Registry*, it starts the *offer analysis* to select the most appropriate option (*offer selection* activity) and to create and register its actual SLA; finally, the API *Consumption* is carried out as long as the API is the *Operation* activity and its regulated based on the terms (such as quotas or rates) defined in the SLA.

In order to implement this lifecycle, it is important to highlight that the *SLA instrumentation*, *SLA enactment* and *Operation* activities should be supported by an SLA enforcement protocol aimed to define the interactions for *checking* if the consumption of the API for a given consumer is allowed (e.g., it meets the limitations specified in its SLA) and to gather the actual values of the *metrics* from the different deployed artifacts that implement the API.

From an industrial perspective and regarding the implication across the entire development lifecycle of APIs, different roles or stakeholders appear, as discussed below. The mapping role-activity is also depicted in Figure 2 by using the RALPH notation [5].

**Developer** This role is composed by the team responsible for the development of a certain API and making it available for other teams. Their use cases are related to the definition of Service Level Objectives (SLOs) since they are the role most aware of the internal functioning of the API. Namely:
- a better understanding of what SLOs can they reasonably target so that they can offer an SLO for the API.
- a better understanding of the performance of their downstream dependencies (e.g., back-ends) so that they can determine their effect on the SLOs.
- a better understanding of the performance of policies in the proxy so that they can determine their effect on the SLOs.

**Product manager** This role is composed of business people, aligned with the company's objectives. Their use cases aim to satisfy customer's needs and be aware of the overall picture of the dependencies between services. Namely, knowing the SLOs of the downstream dependencies so that they can create products which meet the customers' needs.

**Product operator** This role is composed of system administration people, who are responsible for monitoring and reporting the service performance in SLOs. Their use cases aim to be notified of any alert or incident and take remedial actions. Namely:

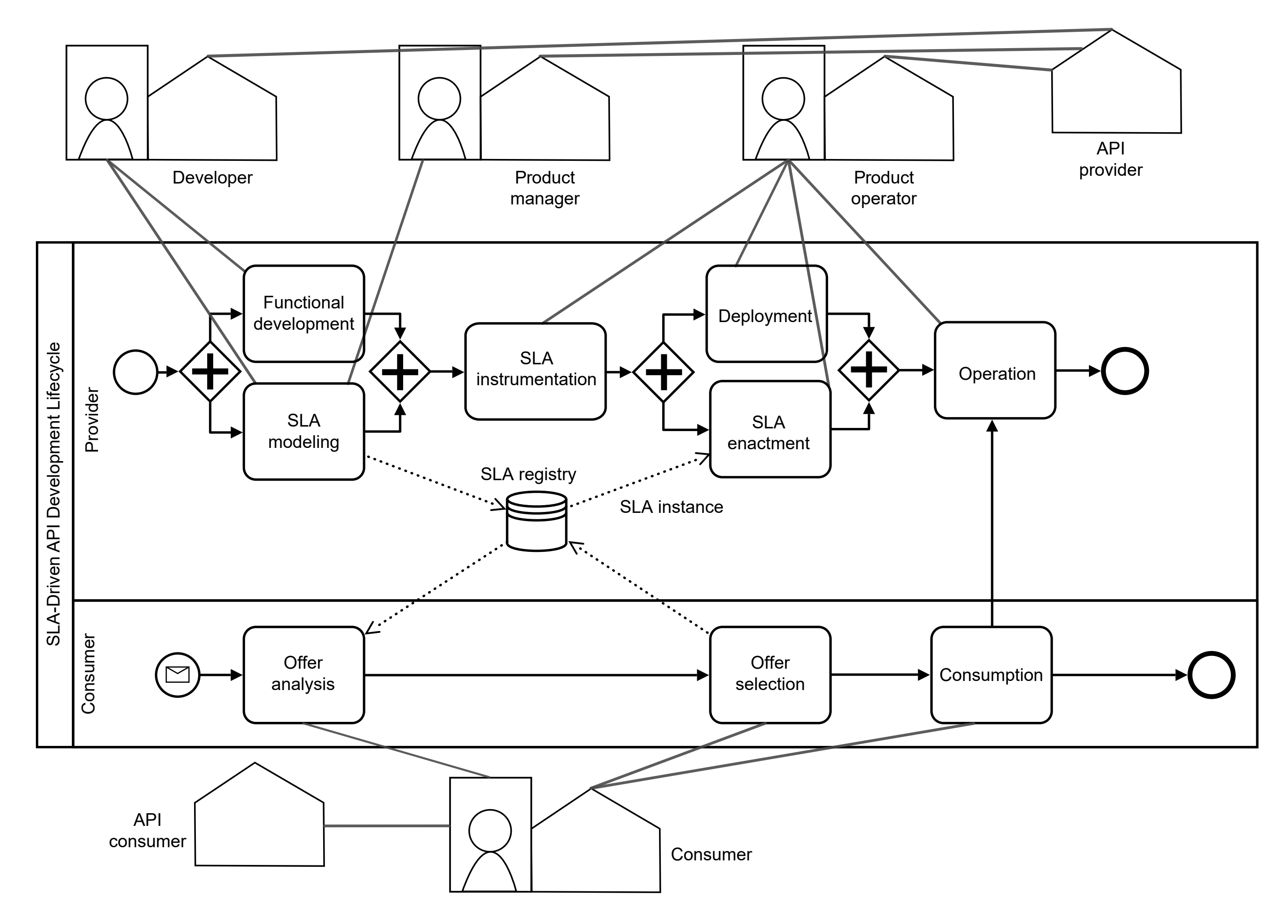



**Figure 2: SLA-Driven API development lifecycle**

- having alerts automatically set based on SLOs to alert them of the risk of missing the objective so that they can take remedial action.
- receiving regular reports detailing API performance against SLOs, so that they can report to the business owners.
- watching both the internal and external SLO commitments for various APIs or Products so that they can quickly categorize and prioritize the operational efforts.

**Consumer** This role is composed of the set of API clients. Their use cases aim to be informed of the different service levels and claim if the SLOs are not being met. Namely:
- knowing what service level is offered so that they can make an informed decision about adopting the API.
- understanding the historical actual performance of an API so that they can know how reliable they might expect them to be.
- assuring that they are getting the service level that they are paying for so that they can claim remedies if SLOs are not met.

# 5 INDUSTRIAL DISCUSSION

## 5.1 The Discussion Process

We opened a call for interest on participating in a research paper open to the OAI members belonging to the SLA4OAI group[7], as part of the OpenAPI Initiative. Our main goal is to gather information about their industrial perspective of the role of SLAs in the APIs.

In order to present general vision, we have classified the participants in different groups regarding their role in the API industry, namely: i) *API infrastructure manager*: are the creator of middleware solutions such as API Gateways or proxies, they do not develop any particular API, but they enhance and enrich third-party ones with other features; ii) *API providers*: are the developers of one or many APIs and also responsible for setting the proper service level and limitations values; iii) *Others*: represent a different set of participant not included before, for instance, API enthusiasts and people who have been involved in the creation of other specifications.

[7] More information at sla@openapi.groups.io

As for an *API infrastructure manager*, we have *Google Apigee*. As for an *API provider*, we have *PayPal*. Finally, *other* participants include *Async API* and *Metadev*.

## 5.2 Describing some API Concepts

In order to have a common vocabulary prior discussion, some considerations about the concepts and terminology took place:

#### 5.2.1 SLA General Concepts.

**Context** describes aspects such as the version, stakeholders or the validity period.

**Metrics** are the elements that are being gathered and computed.

**Service Level Indicator (SLI)** is a particular case of metric which is used to assess one key aspect of the system. They are typically implemented as a time series and may involve some level of sophistication (e.g., sliding windows) in its calculation.

**Service Level Objective (SLO)** is a precise numerical target (often a ratio) for one or more SLIs, describing the minimum acceptable reliability or performance of a system. A given system may have different SLOs for different users, e.g., an internal objective and an external one.

**Guarantee terms** describe the commitments over certain SLI. They also should describe the consequences of not meeting this commitment in terms of compensations.

**Service Level Agreement (SLA)** is, therefore, a contract signed with a user. Notably, SLIs and SLOs are technical constructs whereas SLAs are business constructs.

**Service properties** (or configuration) are the attributes constraints that are being used to drive the API behavior.

#### 5.2.2 API Constraints.

**Quotas** describe the limitations of use for a fixed/static period of time. It is an entitlement to API usage over a (usually relatively long) time period, e.g., 100000 calls per month.

**Rates** describe the limitations of use for a dynamic period of time. It is an entitlement to API usage over a (usually short) time period, e.g., 10 calls per second per consumer.

**Time constraint** some APIs can offer a set of limitations regarding the time in which it is being requested. For instance, some calls could be thought to be cheaper during off-peak hours.

**Authentication** is the verification of the credentials of the request. This process is based on sending the credentials from the remote client to the server by using an authentication protocol. Likewise, the authorization is the process of verification that the connection attempt is allowed. These mechanisms are required for the API monetization.

#### 5.2.3 API Monetization.

**Pricing** is the way in which APIs are monetized. Typically, some pricing models are: fixed (with or without overage) and pay-as-you-go. The first allows a developer to purchase fixed values for a set of metrics (e.g., number of calls) within a period (e.g., per month), but they cannot exceed the established limitations; when overage is allowed, a small fee is charged if the developer exceeds the values of the metrics (e.g., number of calls).

**Plans** is an approach to fit a wide range of business needs by organizing the pricing in a set of tiers of plans.

**Metering** is the recording of the API usage in sufficient detail to perform rating.

**Rating** is the conversion of records of API usage into an owed amount of money. This conversion may involve simply a fixed charge per API call, or considerably more complex schemes.

**Billing** is the presentation to an API user of a report of amounts owed, taking into account any discounts, service credits, taxes, and revenue sharing.

**Collection** is the way of receiving and recording payments of amounts owed by users of APIs.

**Enforcement** is preventing a user from using an API once they have exhausted their pre-paid service credit, or reached a credit limit.

## 5.3 API Provider's Vision

For some API providers, the inclusion of SLAs is something relatively new (less than five years ago), but the main issue is the SLA field is the set of activities surrounding the SLOs to improve the customer experience; for instance, the definitions of metrics and SLIs and the monitoring process.

They believe that, in general, SLOs are drivers for customer experience and digital businesses. As applications and experiences are composed of business capabilities and they are realized as APIs which may use other APIs to achieve their business function, the customer experience is fueled by complex tiered orchestration of APIs and, therefore, performance and availability of experiences is a function of those underlying services.

SLOs for APIs dictate suitability and choice of utilization and, hence, having the ability to accurately measure and monitor SLOs is a fundamental requirement. SLOs, also, dictate performance and availability profiles for the application and provide individual accountability for performance and availability across enabling services. The common thread is the correlation and tracking of the call-chain for service invocation, the identification of the API subscription for applications, monitoring aggregated and apportioned performance profiles for applications and, finally, a common set of performance metrics need to be defined, logged, monitored, analyzed and reported.

As API providers, they use to consider the following set of metrics/SLIs in their APIs:

- **Call volume**: number of API operations invocations irrespective of response.
- **Response time**: the total amount of time, in milliseconds, it takes the service to respond to an API operation request aggregated as the 95th percentile, 90th, and 50th.
- **Availability**: percentage of API calls completed without causing a *Failed Customer Interaction*.
- **Business Error Rate**: percentage of API calls with business error responses. A business error is an error that is not a system error and could be caused by invalid input, user error, business rules, policy constraints, or lack of authorization.
- **System Error Rate**: percentage of API calls with system error responses. A system error is an error that is caused by

a code defect, timeouts for underlying services, or a framework failure, including a hardware network or environment failure.

Regarding the SLI, these metrics need to be measured at the individual API operation level. For REST APIs, the URI for the resource and the HTTP method need to be used as identifiers for API operation. The method identifier from the API specification must be used for correlation. The API operation metrics need to be correlated to the API product and its major and minor version. This correlation will provide insights into capability and ownership attribution form the observed quality of service with respect to published SLAs. The application identity of the originating application, along with that of the immediate application invoking the API operations must be tracked. The identity of Remote Availability Zone (RAZ) for the service application must be tracked to help understand the quality of service across RAZs.

Concerning the monitoring, the published SLOs for API operations must be monitored for compliance. Since there could be variance in API metrics for diverse application use-cases, compliance must be computed using 95th, 90th percentiles, and average aggregations initially, before being base-lined for a longer term. As a daily basis, developer and operation teams are responsible for checking the service status and monitoring the key metrics. Specifically, the SLIs are expected to be in an acceptable range, as defined in the SLOs. For instance, the SLIs availability and latency are measured to meet the target metrics in the SLOs.

Regarding the SLAs, they see SLAs as part of a wider contract, which includes other legal aspects. In such a context, the SLA is just a part of the service contract. At some organizational levels, the value of the SLAs is concentrated in the fulfillment of the guarantee terms when negotiating contractual agreements and invoicing, that is, the SLA reporting. At this point, the SLA of the API services should be considered to be reportable, that is, showing, at a glance, the overall picture of the SLA state in each moment.

In service-based applications (SBAs) the fruitful composition of different services and APIs play a crucial role. There is a strong dependency between different components and, therefore, they are expected to be as reliable as possible (and agreed in the SLA). As an SBA provider, it is strictly necessary to know in advance all the values of the limitations and the agreed SLA terms. Otherwise, the provider is not able to set its own SLOs

## 5.4 API Infrastructure Manager's Vision

As API infrastructure manager, such as an API Gateway, their platforms aim to define API concerns such as different service levels, API limitations (or entitlement) and pricing. They also lay out their position on extending the OpenAPI specification in this area.

Regarding the pricing, their platform provides support for: *fixed fee per API call*, *fixed fee per time period*, *volume-based tiers of fees per API call*, *volume-based bundles of API calls*, *revenue sharing schemes*, *charging variable amounts* based on arbitrary runtime attributes (parameters in the request, elements of the response, time, geography, current load on the API, etc).

They consider two different types of API limitations: quotas and rate limits: i) Quotas are the business level construct of enforcing how much access does one client have to an API based on their tier. For instance: a *gold tier* customer may have access to invoking a set of APIs 1000 per day, whereas a *bronze tier* customer may only able to invoke 100 per day. 2) Rate limiting, on the other hand, has a system-centric connotation. For instance: if the infrastructure is only expected to work for loads under 100 transactions per second, the proper level of rate limiting policy would be irrespective of the kind of customer invoking it.

Regarding the roles, they consider API producers as a team responsible for API development and making the APIs available for every other team. Additionally, they identify the role of an API Product Manager as the one that has business ownership of a portfolio of APIs also known as an API Product. Their main focus is to manage these products and look into ways of monetizing them via partners and external developers. As API infrastructure managers, they use to consider the following set of metrics/SLIs in their APIs:

- **Availability**: percentage of API calls completed without errors.
- **Error rates**: percentage of API calls with error responses
- **Latency**: the total amount of time that takes the service to respond to an API operation request aggregated as a percentile.

Concerning the modeling issues, their current priority would be to codify SLIs and SLOs for APIs in a formal description language by extending the OpenAPI Specification. Based on such a codification their tooling could then offer richer native support for the user stories. Nevertheless, they recommend focusing first on defining an extension to describe technical concerns (e.g., SLIs and SLOs) and keep SLAs (as a business contract) out of the scope for a later extension. They believe that SLAs, as well as not being readily amenable to such a codification, probably don't belong in OpenAPI Specification in any case.

They also suggest that monetization and pricing definition should be part of a separate initiative. In the real world, there is significant complexity in rating API usage, likely deserving of its own OpenAPI extension.

## 5.5 Discussion's Results

In this section, we show some final remarks aiming to be able to define a roadmap in the standardization of the SLA and limitations in an API context.

The relevance of each concept described in Section 5 is different for each provider. After asking them for scoring each one, we gathered and aggregated the responses, as stated in Table 2.

The most important concepts are *metrics/SLIs*, *quotas* and *rates*. The importance of the definition of SLOs for both API producers and infrastructure manager is notorious. As also stated by other participants, it is important to keep separate concerns and different aspects (i.e., SLOs, plans, metrics); they can be always be referenced externally if needed. The granularity of definitions when defining an SLA model is a problem: there exists the dichotomy between a fine-grained approach (i.e., a fully comprehensive model description) and a coarse-grained one (i.e., a description the most common elements and paving the way for custom extensions).

**Table 2: Relevance of concepts for industrial participants**

| Items | | Score |
|---|---|---|
| General concepts | Context | ●●○ |
| | Metrics | ●●● |
| | SLIs | ●●● |
| | SLOs | ●●○ |
| | Guarantees | ●●○ |
| | SLAs | ●○○ |
| | Configuration | ○○○ |
| API constraints | Quotas | ●●● |
| | Rates | ●●● |
| | Time constraints | ●○○ |
| | Authorization | ●○○ |
| API monetization | Pricing | ●○○ |
| | Plans | ●○○ |
| | Metering | ●●○ |
| | Rating | ●●○ |
| | Billing | ●○○ |
| | Collection | ●○○ |
| | Enforcement | ●●○ |

Symbol ● denotes the relevance for the industrial participants.

In general terms, the participants belonging to the SLA4OAI group, as part of the OpenAPI Initiative, tend to agree in a manifesto during the standardization tasks:

**Motivation** fostering the importance of the SLA inside the API development lifecycle is that SLAs are already present in most commercial APIs. Since OAI is becoming the *de facto* standard for the definition of APIs, natural evolution to describe SLAs into OpenAPI Specification would expand the OAI benefits.

**Goals** Three are identified:
- Be as aligned as possible with the OpenAPI principles.
- Describe the most common elements in SLAs (e.g., plans, metrics, quotas, rates).
- Be integrated with the main OpenAPI Specification.

**Non-goals** There are two:
- Define a particular way to enforce SLAs.
- Be fully comprehensive including a wide set of elements found in different industrial APIs.

**Design principles** They are two:
- Pragmatism to spot the most common elements;
- Promote tooling to take advantage of the SLA4OAI Specification.

## 6 CONCLUSIONS

From the Academia's point of view, the fact of having a standard model for the definition of SLAs in APIs could foster the development of novel techniques aiming to deal with the information contained in the SLAs. There is already a number of works in the SLA field, as pointed out in Section 3, so aligning that with the API ecosystem would pave the way for new challenges.

As an example, this SLA model could enable *SLA-aware monitoring and testing* techniques: including non-functional and QoS requirements into the test cases. Moreover, a *formal analysis on the SLA model* could unveil inconsistencies in the set of API limitations. Furthermore, *SLA-aware model-driven development* would experience an improvement, since taking into account the SLA could be helpful when deciding among different architectures. A first step in this direction, in [9], we presented *Governify for APIs*, an initial set of tools aimed to settle down our idea of SLA-driven APIs.

Finally, this work is intended to collect the industrial perspective on the challenge of standardizing the modeling of SLAs and limitations in the API context, under the umbrella of a well-assented specification for APIs as it is the OpenAPI Specification. The contribution presented herein just lay the first stone on the roadmap that is the modeling effort in conjunction with relevant industrial players.

## ACKNOWLEDGMENTS

This work is partially supported by the European Commission (FEDER), the Spanish Government under projects BELI (TIN2015-70560-R) and HORATIO (RTI2018-101204-B-C21), and the FPU scholarship program, granted by the Spanish Ministry of Education, Culture and Sports (FPU15/02980).

The authors would also like to thank for their time and their valuable contributions to all the members of the OpenAPI Technical Steering Committee and, specially, to the rest of the Technical Committee behind the SLA4OAI Specification: Isaac Hepworth (Google), Jeffrey ErnstFriedman (The Linux Foundation), Kin Lane (API Evangelist), Mike Ralphson (The Linux Foundation) and Scott Ganyo (Google).

## REFERENCES

[1] Mithun Acharya, Tao Xie, Jian Pei, and Jun Xu. 2007. Mining API patterns as partial orders from source code. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07*. ACM Press, New York, New York, USA, 25. https://doi.org/10.1145/1287624.1287630

[2] Alain Andrieux, Karl Czajkowski, Kate Keahey, A. Dan, Kate Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. 2004. Web Services Agreement Specification (WS-Agreement). (2004), 80 pages. http://forgc.gridforum.org/Public_Comment_Docs/Documents/Oct-2006/WS-AgreementSpecificationDraftFinal_sp_tn_jpver_v2.pdf

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems* (1st ed.). O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA.

[4] Michele Bonardi, Maurizio Brioschi, Alfonso Fuggetta, Emiliano Sergio Verga, and Maurilio Zuccalà. 2016. Fostering Collaboration Through API Economy: The E015 Digital Ecosystem. In *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice (SER&IP '16)*. ACM, New York, NY, USA, 32–38. https://doi.org/10.1145/2897022.2897026

[5] Cristina Cabanillas, David Knuplesch, Manuel Resinas, Manfred Reichert, Jan Mendling, and Antonio Ruiz-Cortés. 2015. RALph: A Graphical Notation for Resource Assignments in Business Processes. In *Advanced Information Systems Engineering*, Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson (Eds.). Springer International Publishing, Cham, 53–68.

[6] Roy Thomas Fielding. 2000. Architectural Styles and the Design of Network-based Software Architectures. *Building* 54 (2000), 162. https://doi.org/10.1.1.91.2433

[7] Forrester. 2015. *API Management Solutions , Q3 2014*. Technical Report. Forrester.

[8] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. 2017. An Analysis of RESTful APIs Offerings in the Industry. In *Service-Oriented Computing*, Michael Maximilien, Antonio Vallecillo, Jianmin Wang, and Marc Oriol (Eds.). Springer International Publishing, Cham, 589–604.

[9] Antonio Gamez-Diaz, Pablo Fernandez, and Antonio Ruiz-Cortes. 2019. Governify for APIs: SLA-Driven ecosystem for API governance. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the*

*Foundations of Software Engineering (ESEC/FSE 2019)*. ACM, Tallin, Estonia. https://doi.org/10.1145/3338906.3341176

[10] Antonio Gámez-Díaz, Pablo Fernández-Montes, and Antonio Ruiz-Cortés. 2015. Towards SLA-Driven API Gateways. In *Actas de las XI Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios*, Juan Manuel Murillo (Ed.), Vol. 201232273. Sistedes, Santander, 9. https://doi.org/10.13140/RG.2.1.4111.5609

[11] José María García, Pablo Fernández, Carlos Pedrinaci, Manuel Resinas, Jorge Cardoso, and Antonio Ruiz-Cortés. 2017. Modeling Service Level Agreements with Linked USDL Agreement. *IEEE Transactions on Services Computing* 10, 1 (1 2017), 52–65. https://doi.org/10.1109/TSC.2016.2593925

[12] Holger Harms, Collin Rogowski, and Luigi Lo Iacono. 2017. Guidelines for Adopting Frontend Architectures and Patterns in Microservices-based Systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 902–907. https://doi.org/10.1145/3106237.3117775

[13] Keven T. Kearney, Francesco Torelli, and Constantinos Kotsokalis. 2010. SLA * An abstract syntax for Service Level Agreements. In *2010 11th IEEE/ACM International Conference on Grid Computing*. IEEE, Brussels, Belgium, 217–224. https://doi.org/10.1109/GRID.2010.5697973

[14] Yousri Kouki, Frederico Alvares de Oliveira, Simon Dupont, and Thomas Ledoux. 2014. A language support for cloud elasticity management. In *Proceedings - 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing, CCGrid 2014*. IEEE, Chicago, IL, USA, 206–215. https://doi.org/10.1109/CCGrid.2014.17

[15] D. D. Lamanna, J. Skene, and W. Emmerich. 2003. SLAng: A language for defining service level agreements. In *FTDCS*, Vol. 2003-Janua. IEEE, San Juan, Puerto Rico, USA, USA, 100–106. https://doi.org/10.1109/FTDCS.2003.1204317

[16] H. Ludwig, A. Keller, A. Dan, and R. King. 2002. A service level agreement language for dynamic electronic services. In *WECWIS 2002*. IEEE Comput. Soc, Newport Beach, CA, USA, USA, 25–32. https://doi.org/10.1109/WECWIS.2002.1021238

[17] C. Muller, A. Gutierrez Fernandez, P. Fernandez, O. Martin-Diaz, M. Resinas, and A. Ruiz-Cortes. 2018. Automated Validation of Compensable SLAs. *IEEE Transactions on Services Computing* (jan 2018), 1–1. https://doi.org/10.1109/TSC.2018.2885766

[18] Andy Neumann, Nuno Laranjeiro, and Jorge Bernardino. 2018. An Analysis of Public REST Web Service APIs. *IEEE Transactions on Services Computing* (2018). https://doi.org/10.1109/TSC.2018.2847344

[19] Tien N. Nguyen, Anh Tuan Nguyen, Trong Nguyen, Thanh Nguyen, Hoan Anh Nguyen, Ngoc Tran, Hung Phan, and Linh Truong. 2018. Complementing global and local contexts in representing API descriptions to improve API retrieval tasks. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press, New York, New York, USA, 551–562. https://doi.org/10.1145/3236024.3236036

[20] Rick Rabiser, Sam Guinea, Michael Vierhauser, Luciano Baresi, and Paul Grünbacher. 2017. A comparison framework for runtime monitoring approaches. *Journal of Systems and Software* 125 (3 2017), 309–321. https://doi.org/10.1016/j.jss.2016.12.034

[21] Anastasia Reinhardt, Tianyi Zhang, Mihir Mathur, and Miryung Kim. 2018. Augmenting Stack Overflow with API Usage Patterns Mined from GitHub. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 880–883. https://doi.org/10.1145/3236024.3264585

[22] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar. 2016. From the Service-Oriented Architecture to the Web API Economy. *IEEE Internet Computing* 20, 4 (July 2016), 64–68. https://doi.org/10.1109/MIC.2016.74

[23] S. Tata, M. Mohamed, T. Sakairi, N. Mandagere, O. Anya, and H. Ludwig. 2016. rSLA: A Service Level Agreement Language for Cloud Services. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, San Francisco, CA, USA, 415–422. https://doi.org/10.1109/CLOUD.2016.0062

[24] Rafael Brundo Uriarte, Francesco Tiezzi, and Rocco De Nicola. 2014. SLAC: A Formal Service-Level-Agreement Language for Cloud Computing. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC '14)*. IEEE Computer Society, Washington, DC, USA, 419–426. https://doi.org/10.1109/UCC.2014.53